

# Algorithms for Deterministic Call Admission Control of Pre-stored VBR Video Streams

Christos Tryfonas\*

Kazeon Systems, Inc., 1161 San Antonio Road , Mountain View, CA 94043, USA  
Email: tryfonas@kazeon.com

Dimitris Papamichail\*

Computer Science Department, University of Miami, Coral Gables, FL 33146, USA  
Email: dimitris@cs.miami.edu

Andrew Mehler, Steven S. Skiena

Computer Science Department, SUNY at Stony Brook, Stony Brook, NY 11794, USA  
Email: {mehler, skiena}@cs.sunysb.edu

**Abstract**—We examine the problem of accepting a new request for a pre-stored VBR video stream that has been smoothed using any of the smoothing algorithms found in the literature. The output of these algorithms is a piecewise constant-rate schedule for a Variable Bit-Rate (VBR) stream. The schedule guarantees that the decoder buffer does not overflow or underflow. The problem addressed in this paper is the determination of the minimal time displacement of each new requested VBR stream so that it can be accommodated by the network and/or the video server without overbooking the committed traffic. We prove that this call-admission control problem for multiple requested VBR streams is NP-complete and inapproximable within a constant factor, by reducing it from the VERTEX COLOR problem. We also present a deterministic morphology-sensitive algorithm that calculates the minimal time displacement of a VBR stream request. The complexity of the proposed algorithm along with the experimental results we provide indicate that the proposed algorithm is suitable for real-time determination of the time displacement parameter during the call admission phase.

**Index Terms**—Variable Bit-Rate Stream, Call-Admission Control, Time Displacement, 3SUM hard, Constant Factor Inapproximable.

## I. INTRODUCTION

A significant portion of the forecasted network traffic is expected to be multimedia (e.g. voice and video) traffic. New services such as video-on-demand (VoD) and TV broadcasting are currently under massive deployment. One of the salient characteristics of video traffic is that it usually exhibits high variability in its bandwidth demands in different time scales. The need to better understand the bandwidth demands of video streams is essential for

proper resource provisioning of both the network resources and the resources of the video servers when stored video is transported. Proper resource dimensioning has direct correlation with the quality of the recovered video on the decoder and, therefore, a variety of techniques have been proposed in the past.

Significant work has been done in the literature in the area of statistical modeling of video traffic for resource provisioning purposes, so that it can be effectively transported over packet-switched networks [1]–[5]. In most cases, the objective of these efforts is to build a general model that can be used for resource dimensioning for all the video traffic transported over the network. In some cases, the long-range dependence (LRD) characteristic of video traffic is exploited to create a model of the traffic source [1], [6], [7]. These methods, in general, characterize the traffic source based on its statistical properties, and provide value when the video stream is not known a-priori. However, when dealing with pre-stored video, the resource dimensioning process can be made deterministic and any statistical technique is of limited value, since it does not capture the exact dynamics of the video stream in the time domain.

In video applications that transport stored video over a packet-switched network, the resource provisioning process can take advantage of the fact that video streams can be pre-processed off-line. During the pre-processing of a video stream, a transmission schedule is typically computed to minimize its rate variability and, therefore, facilitate the resource provisioning and the call admission control process. The reduction in rate variability is done by work-ahead smoothing, i.e. sending more data to the receiver with respect to its playback time. Significant

\*Authors contributed equally to this work.

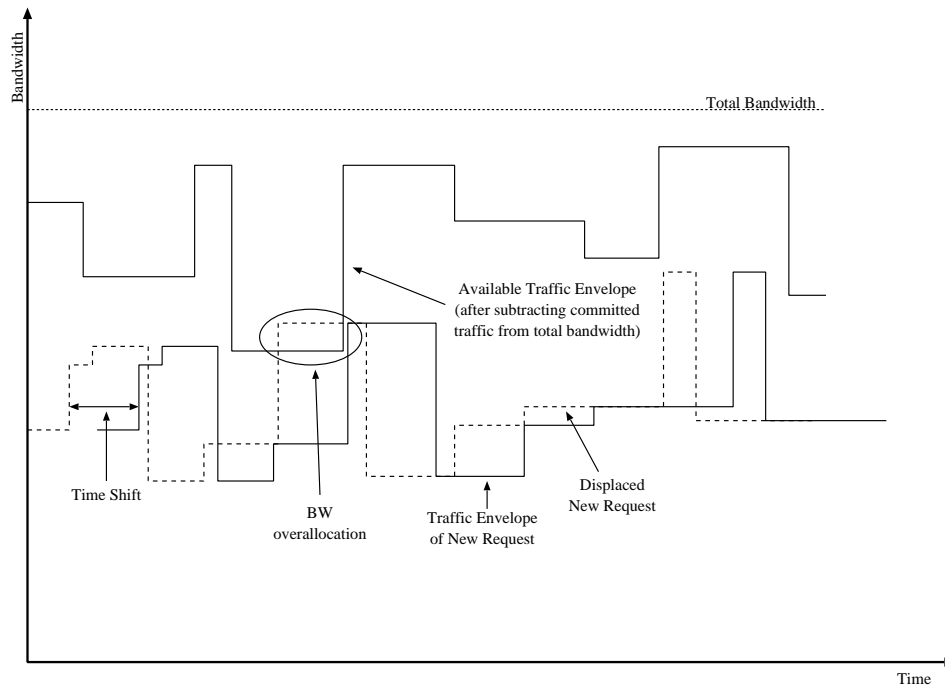


Figure 1. Example of accommodating a new video call request through time displacement.

work can be found in the literature in the area of work-ahead video smoothing [8]–[11]. The general idea behind most of these algorithms is to maximize the time intervals (rate segments) at which a transmission rate for the video stream is used without causing under/overflow of the receiver buffer. The algorithms differ in the selection of the starting point of these rate segments. The output of these algorithms is a piecewise constant-rate schedule for the smoothed video stream. The schedule guarantees that the decoder buffer does not overflow or underflow. Due to the fact that computing the smoothing schedule is not a trivial process and cannot be performed online, the pre-computed smoothed schedules of a video file for various decoder profiles can be stored along with the file itself in the video servers, so that they can be used at the time of the corresponding video request to guarantee a deterministic quality at the decoder.

The problem addressed by this paper is that of accepting a new request for a pre-stored VBR video stream that has been pre-smoothed using any of the smoothing algorithms. Since the request can come at any particular point in time, the problem is related to the accommodation of the new request provided that the envelope of the dynamics of the committed traffic, and therefore the envelope of the available bandwidth in the channel, does not introduce over-allocation at any time interval (see Fig. 1). The goal is to displace the pre-computed smoothed schedule of the new request into the future to avoid over-allocation. More specifically, we want to find the minimum time displacement of the new schedule so

that the channel can accommodate the new request. The problem described is an optimization problem that can be extended in several ways. For example, given a set of requests, find the displacement points of the associated schedules so that the overall schedule is the smoothest.

In this paper we present an algorithm that solves the problem of computing the minimum displacement of a new request, henceforth referred to as the TWO STREAM SCHEDULING problem (2-SS). This is a morphology-sensitive algorithm with  $O((P+n)\log n)$  complexity, where  $n$  is the size of the input and  $P = O(n^2)$  the number of possible locations where bandwidth over-allocation can occur. This algorithm has  $O(n^2\log n)$  time complexity as a function of the input size alone and makes specific observations about the smoothed schedule so that certain peaks can be skipped by the algorithm to speed-up the final calculation considerably, depending on the input. Then we present a lower bound on the complexity of the 2-SS problem, which is shown to belong in the 3SUM-hard problem group. We also demonstrate that the problem of computing the minimum displacement of multiple new requests (also referred to as the MULTIPLE STREAM SCHEDULING problem or  $m$ -SS) is NP-complete, and cannot be polynomially approximated within a constant factor. This is proven by reducing the STRING PACK problem to  $m$ -SS. STRING PACK was introduced and shown NP-complete in [12]. To obtain the approximability results, we further reduce VERTEX COLOR [13] to STRING PACK. This reduction yields new hardness of approximability bounds for STRING PACK,

thus improving previous results.

The rest of this paper is organized as follows: In Section II, we present the formal definition of the problem for call admission of two VBR streams (or equivalently the admission control of a new request over the envelope of available bandwidth in a channel). We also propose an algorithm that is efficient for the 2-SS problem. In Section III, we extend the problem to multiple streams. We prove that the problem of admitting multiple streams is NP-complete by reducing the STRING PACK problem to it. We present experimental results that demonstrate the performance of the algorithm in practical scenarios in Section IV. Finally, in Section V we conclude the paper with a summary of this work and possible extensions.

## II. ADMISSION CONTROL OF A NEW REQUEST

### A. Formal definition

The input of the TWO STREAM SCHEDULING (2-SS) problem is two ortholinear traffic envelopes (streams)  $S_1$  and  $S_2$  of total length  $L_1$  and  $L_2$  respectively and the channel bandwidth  $B$ . A stream envelope  $S_k$ ,  $k \in \{1, 2\}$ , can be described by an ordered set of triplets  $r_{ki} = (h_{ki}, s_{ki}, e_{ki})$ ,  $i = 1 \dots n$ , with  $h_{ki}$  being the height value (bandwidth demand of video) and  $s_{ki}$  and  $e_{ki}$  the starting and ending time points of the  $i$ th peak respectively, of a total of  $n$  non-overlapping peaks in the stream. Let  $l_{ki} = e_{ki} - s_{ki}$  be the length of the  $i$ th peak. For the 2-SS problem,  $S_1$  consists of  $n$  such triplets and  $S_2$  of  $m = O(n)$  triplets.

We consider  $S_1$  being requested and transmitted at time point  $s_{10} = 0$ , so being fixed at that position. This allows us to subtract its content allocation from the total bandwidth, creating a reverse envelope, as in Fig. 1. Basically, stream  $S_1$  corresponds to the committed traffic. The second stream can be displaced by a positive time interval  $T$  to its right, resulting in delayed transmission. We assume that the envelopes are rigid and none of the peaks can be altered either in length or height. The order of the peaks is fixed.

Let  $S_2$  be displaced by  $T \geq 0$  time units. An intersection (time overlap) of the  $r_{1i} = (h_{1i}, s_{1i}, e_{1i})$  peak triplet from  $S_1$  with the  $r_{2j} = (h_{2j}, s_{2j}, e_{2j})$  peak triplet from  $S_2$  occurs when  $h_{1i} + h_{2j} > B$  and  $\exists t \in [s_{1i}, e_{1i}] : t \in [s_{2j} + T, e_{2j} + T]$ . The set of all time points such that  $r_{1i}$  intersects  $r_{2j}$  defines a *time interval* (referred to from now on as *intersection interval* or just *interval*)  $t_{ij} = [T_{ij1}, T_{ij2}]$  of length  $l_{1i} + l_{2j}$ , starting at time point  $T_1 = s_{1i} - e_{2j}$  and ending at  $T_2 = e_{1i} - s_{2j}$ . Intersection parameters are depicted graphically in Fig. 2. The second stream cannot be displaced by any value corresponding to this intersection interval, or there will occur a bandwidth over-allocation.

The output of the 2-SS algorithm will be the minimum displacement  $T_{min}$  of  $S_2$ , such that there is no bandwidth over-allocation. The second stream can be shifted only by a displacement that does not fall into any intersection interval  $t_{ij}$ , for  $1 < i < n$  and  $1 < j < m$ . So, the output of the algorithm could be described as the minimum displacement that does not fall into an intersection interval.

### B. A morphology sensitive algorithm

In this section we describe a morphology sensitive algorithm to solve the 2-SS over-allocation problem. That is an algorithm whose asymptotic time complexity depends on the morphology of the input as well as its size. The algorithm processes all segments, in order to calculate their intersection intervals. It could be the case though that many peaks will not be as high as to intersect. By sorting the peaks by height (bandwidth demand), one can actually calculate the intersection intervals only for the ones that actually intersect and not consider the rest.

Let  $P$  be the number of peak pairs, where the first peak is selected from envelope  $S_1$  and the second from envelope  $S_2$ , that have sum of heights greater than the bandwidth  $B$  and thus define an intersection interval. The algorithm then goes as follows:

- 1) Sort the peak triplets of both envelopes according to height.
- 2) Iterate through sorted peaks in  $S_1$  and calculate their intersection interval with all peaks from the sorted list of  $S_2$  that cause bandwidth over-allocation. Stop when the height of the next peak in  $S_1$  does not intersect the highest peak of  $S_2$ .
- 3) Sort all intersection intervals according to their starting point.
- 4) Iterate through sorted intersection intervals, merging them into an *aggregate interval*

$$I = [T_s, T_e] = \bigcup_{k=1}^i t_k$$

where  $t_i$  is the currently processed intersection interval, until an interval  $t_i$  that does not intersect with the aggregate interval is discovered ( $I \cap t_i = \emptyset$ ), or we run out of intervals.

- 5) Output the end point of the aggregate interval as the solution  $T_{min}$ , unless the aggregate interval starts at  $T_s \geq 0$  or no intersection interval exists, in which case output  $T_{min} = 0$ .

For the correctness of the algorithm we can argue that by iterating through all intersecting peaks of both streams, we have discovered all possible time intervals where the second stream cannot be shifted. The first "gap" between the aggregate interval and the currently

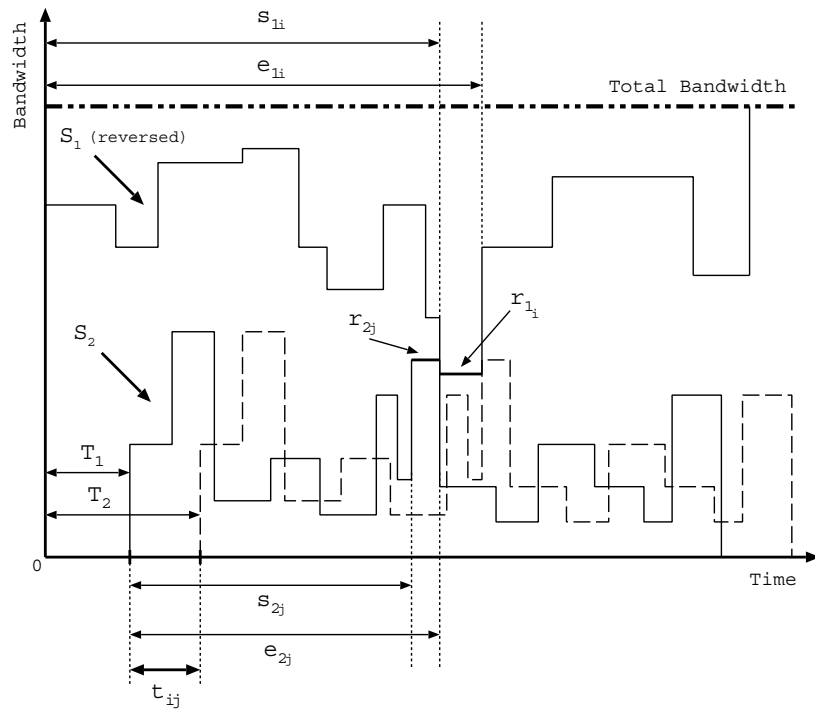


Figure 2. Intersection interval parameter display.

examined interval will provide the minimum displacement, since any position in the aggregate interval defines a forbidden displacement, belonging to some previously examined intersection interval. The start of the “gap” described above cannot belong to any interval, since, if such an interval existed, its start would occur before the end of the aggregate interval and, as such, before the currently examined interval, which means it would have already been included in the aggregate interval. A visual representation of the procedure can be seen in Fig. 3.

Sorting the peaks by height requires  $O(n \log n)$  time, the iteration through sorted peaks takes  $O(P)$  time (where  $P = O(n^2)$  is the number of intersecting peak pairs that can cause bandwidth over-allocation) and sorting all intersection intervals takes  $O(P \log P)$  time. The merging iteration takes at most  $O(P)$  time. So the total complexity is  $O((P + n) \log n)$ .

It should be noted that in the worst case scenario where the number of intersecting peaks between the two streams is  $\Theta(n^2)$ , the asymptotic complexity becomes  $O(n^2 \log n)$ , dominated by sorting the intersection intervals’ starting points. We can further improve the running time of this algorithm by excluding intersection interval calculation for peak pairs that result in negative second stream displacement, although such an optimization does not result in any asymptotic gain.

C. 2-SS scheduling is 3SUM hard

In this section we will prove that 2-SS is 3SUM-hard, a class of problems introduced in [14]. The 3SUM problem is to decide whether there exist integers  $a, b, c$  in a set of  $n$  integers, such that  $a + b + c = 0$ , which is currently considered to have complexity  $\Theta(n^2)$ .

The notion of 3SUM-hardness (or  $n^2$ -hardness) is formally introduced in [14], [15], the notation of which we follow. In brief, we will mention that a problem is considered 3SUM-hard if any instance of the 3SUM problem can be reduced to some instance (with a comparable size) of the other problem in  $o(n^2)$  time, where  $n$  is the size of the input.

For our proof, we will need the following definition:

*Definition 1:* Given two problems PR1 and PR2 we say that PR1 is  $f(n)$ -solvable using PR2 if every instance of PR1 of size  $n$  can be solved by using a constant number of instances of PR2 (of size  $O(n)$ ) and  $O(f(n))$  additional time. We denote this by

$$PR1 \lll_{f(n)} PR2$$

To prove that 2-SS is 3SUM-hard, it will be sufficient to show that another 3SUM-hard problem is  $o(n^2)$ -solvable using 2-SS. For that purpose, we will use the following 3SUM-hard problem:

**Problem:** SCP (Segments Containing Points): Given a set  $P$  of  $n$  real numbers and a set  $Q$  of  $m = O(n)$  pairwise-disjoint intervals of real numbers, is there a real number translation  $u$  such that  $P + u \subseteq Q$ ? Here,  $P + u$

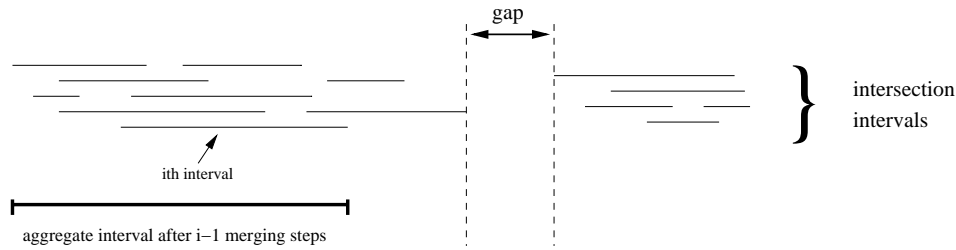


Figure 3. Merging intersection intervals into an aggregate interval at the  $i - 1$  step. The start of the gap defines the  $S_2$  stream displacement where no bandwidth over-allocation occurs.

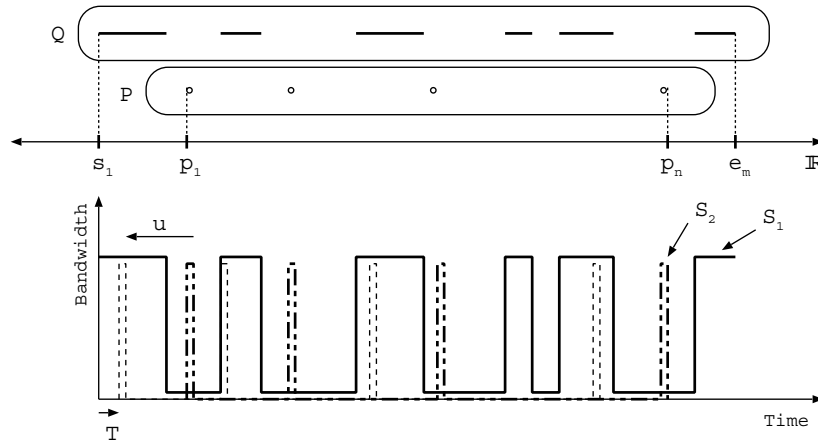


Figure 4. 2SS construct.  $S_1$  was translated in height for better viewing.

indicates the set of intervals in  $P$  translated by  $u$ .

$SCP$  was shown  $3SUM$ -hard in [15]. We will now prove the following:

*Theorem 1:*  $SCP \lll_{n \log n} 2 - SS$

*Proof:* Given an instance of the  $SCP$  problem, we construct two streams in the following way: The  $m$  intervals in set  $Q$  and  $n$  real numbers in set  $P$  are sorted and stream  $S_1$  is constructed to have peaks of height 0 on these intervals and peaks of height 1 in between. The length  $L_1$  of  $S_1$  is determined by the start of the first interval  $s_1$  and end of the last interval  $e_m$  in the sorted list and starts at time point 0, with every segment displaced in time by subtracting  $s_1$  from each of its coordinates. Stream  $S_2$  is constructed with peaks of length  $\varepsilon$  with  $\varepsilon \rightarrow 0$  of height 1 at locations defined by the sorted numbers of set  $P$ , with peaks of height 0 in the intervals in between. The length  $L_2$  of  $S_2$  is again determined from the smallest and largest elements of  $P$  ( $p_1$  and  $p_n$  respectively) and original displacement  $T$  of 0 is achieved by subtracting  $p_1$  from all peak segment coordinates. We set the channel bandwidth  $B = 1$ . The construct can be seen in Fig. 4.

We will now argue that the instance of  $SCP$  has a solution if and only if the corresponding instance of 2-SS has a displacement solution less than  $L_1 - L_2$  (if  $L_2 > L_1$  there is no solution). From the construction it is obvious that a peak of  $S_2$  of height 1 can fit under a 0 height

peak of stream  $S_1$  only if the corresponding number in  $P$  falls in the corresponding interval of  $Q$ . If for a certain displacement  $T$  of  $S_2$  we have  $T < L_1 - L_2$  and there is no over-allocation of bandwidth, then all peaks of  $S_2$  of height 1 fit under 0-height peaks of stream  $S_1$ , which would imply that  $\exists u = T + s_1 - p_1 : P + u \subseteq Q$ . Also, by the same arguments, if there  $\exists u : P + u \subseteq Q$ , then  $S_2$  displaced by  $T = u + p_1 - s_1$  will result in scheduling the two streams with no over-allocation. ■

Based on this result, we can conclude that our morphology-sensitive algorithm for scheduling two streams is within a log factor from optimality, considering the 2-SS problem time complexity.

### III. SCHEDULING MULTIPLE STREAMS

We now extend the 2-SS problem to MULTIPLE STREAM SCHEDULING ( $m$ -SS), where the input would consist of multiple VBR streams that we want to schedule for transmission over a fixed bandwidth channel. Although we could set different objectives for optimization, we will select minimizing the displacement of the last stream being transmitted. For streams of the same size this is equivalent to minimizing the total length of transmission, starting from the time point of the first stream transmission and ending when the last stream has been transferred over the channel.

**A. Multi-stream scheduling is NP-complete**

To demonstrate that  $m$ -SS is NP-complete, we will reduce the STRING PACK problem to it. The STRING PACK problem appeared in [12] and was proved hard by reduction from 3-PARTITION.

The STRING PACK is defined as follows: Given a set of  $m$  strings of length  $n$ , over the binary alphabet  $\Sigma = \{0, 1\}$ , find a minimum length  $l$  packing (alignment) of the strings, such that no column has more than one '1'. An example of the input and the output of the problem are shown in Fig. 5.

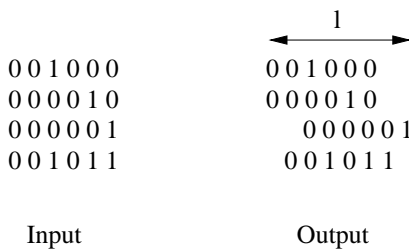


Figure 5. STRING PACK example with  $m = 4, n = 6$  and  $l = 8$

The reduction is straightforward. We will transform the input binary strings into streams with peaks of height 1 for each '1' encountered in the string and peaks of height 0 for each '0' appearing in the string, as shown in Fig. 6

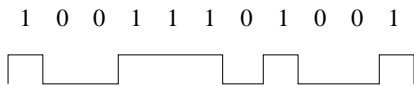


Figure 6. String and equivalent stream transformation

We let the total available bandwidth  $B = 1$ , such that no peaks from any stream can overlap. This adheres to the requirement of the STRING PACK problem not having any column with more than one '1'. Since the input to the STRING PACK problem is a set of strings with equal length  $n$ , minimizing the total length of the outputted alignment is equivalent to minimizing the displacement of the last string. Thus, the output of the  $m$ -SS on the transformed strings provides that exact minimum. So we have the following:

**Theorem 2:**  $\text{STRING PACK} \leq_p m\text{-SS}$

*Proof:* Given an instance of STRING PACK, create an  $m$ -SS instance by transforming the binary strings to equivalent streams as described above. The minimum displacement of the last stream to be transmitted, added to the length  $n$  of the strings, provides the minimum length of the  $m$  strings' packing. ■

The result that  $m$ -SS is NP-complete follows from the observation that given a string packing, it can be verified in time  $O(mn)$  (thus polynomial in the input length) that it constitutes a valid solution, where no column in the packing has more than one '1', and that the length of the

packing is less than a specified length  $k$ , which would be an input of the decision version of the problem.

**B. MULTIPLE STREAM SCHEDULING is polynomially inapproximable within a constant**

VERTEX COLOR is a well known problem [13], defined as follows: Given a graph  $G = (V, E)$ , color the vertices of  $V$  with the minimum number of colors such that for each edge  $(i, j) \in E$ , vertices  $i$  and  $j$  have different colors.

It has been shown that VERTEX COLOR is inapproximable within  $|V|^{1-\epsilon}$  for any  $\epsilon > 0$ , unless  $Zpp = NP$  [16]. By reducing VERTEX COLOR to STRING PACK and with the reduction of the latter to  $m$ -SS, shown in the previous section, we will demonstrate that any constant approximation of  $m$ -SS is NP-hard.

We now proceed showing that VERTEX COLOR reduces to STRING PACK.

Consider a graph  $G = (V, E)$ , and its vertex-edge incidence matrix. As a running example, we will use the graph given in (Fig. 7) whose incidence matrix is shown below.

$$\begin{matrix}
 & \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\
 v_1 & \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\
 v_2 & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\
 v_3 & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\
 v_4 & \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}
 \end{matrix}$$

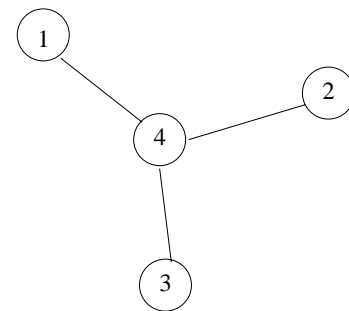


Figure 7. A Graph on 4 vertices.

It is clear that the graph can be colored with 2 colors.  $v_4$  gets one color, and  $\{v_1, v_2, v_3\}$  get another color. Also, the rows of the incidence matrix corresponding to a color group can all be packed with no collision. For example, putting together the rows for  $\{v_1, v_2, v_3\}$  gives

$$\begin{matrix}
 & \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\
 v_1 & \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\
 v_2 & \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\
 v_3 & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\
 \text{Sum} & \begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}
 \end{matrix}$$



TABLE II.  
SELF-ALIGNING STRINGS' EXAMPLES

(a) First attempt at self-aligning strings														
1	(	1111	0000	0000	0000	1000	1000	1000	1000	1000	1000	...	)	
2		0000	1111	0000	0000	0100	0100	0100	0100	0100	...			
3		0000	0000	1111	0000	0010	0010	0010	0010	0010	...			
4		0000	0000	0000	1111	0001	0001	0001	0001	0001	...			
(b) Second attempt at self-aligning strings														
1	(	0000	1111	0000	0000	1111	0000	0000	0000	1000	1000	1000	1000	...
2		0000	0000	1111	0000	0100	0100	0100	0100	0100	...			
3		0000	0000	1111	0000	0010	0010	0010	0010	0010	...			
4		0000	0000	0000	1111	0001	0001	0001	0001	0001	...			

be  $n$  consecutive 1's followed by repeated identity matrices, as shown in Table II(a).

The strings are grouped for clarity. In the first 4 blocks, each row gets a sequence of 4 consecutive 1's. The rest of the blocks are identity matrices. In the region with the identity matrix, any sub-matrix of 4 consecutive columns is a permutation matrix (ie each row has a 1 in it). Thus, once the 4-consecutive 1's are shifted into this region, they will always collide with every string.

However, we see these are not self-aligning strings, since the consecutive 1 blocks must be shifted by as much as 16 places before they are in the identity matrix region of the other strings, as seen in Table II(b).

To prevent shifts of 1 to 15 we can add the following types of strings to the end of the above strings.

$$\begin{array}{r} 1 \\ 2 \\ 3 \\ 4 \end{array} \left( \begin{array}{cccc} 1000 & 0000 & 0000 & 0000 \\ 0111 & 1111 & 1111 & 1111 \\ 0000 & 0000 & 0000 & 0000 \\ 0000 & 0000 & 0000 & 0000 \end{array} \right)$$

This matrix prevents the first row from shifting an amount 1 to 15 with the second row. We concatenate strings like these for every pair of rows ( $4^2 = 16$ ). The final idea in this construction is that there is no limit on the number of identity matrices we included. Thus, we can make these strings as long as we need, until the allowed overlap is a small enough fraction (for example, add  $n^{100}$  identity matrices). For a more precise explanation of self-aligning strings, consult the Appendix.

We now proceed to prove the following:

**Theorem 3:** VERTEX COLOR  $\leq_p$  STRING PACK

*Proof:* Given an instance of VERTEX COLOR, create a STRING PACK instance with the vertex-edge incidence matrix flanked by self-aligning strings. The number of groups in the solution to STRING PACK is the number of colors in an optimal coloring. ■

**Theorem 4:** STRING PACK is hard to approximate (No constant factor approximation).

*Proof:* We can approximate VERTEX COLOR with STRING PACK. The approximation depends on the length

of the flanking regions. As described in the appendix, we construct flanking strings of size  $O(n^5)$ . Thus the total size of the STRING PACK instance is  $O(n^6)$ . Since  $n$  is the number of vertices, the size of VERTEX COLOR problems are  $O(m) = O(n^2)$ . So if we have an  $f(n)$  approximation to STRING PACK, we get an  $f(n^6) = f(m^3)$  approximation to VERTEX COLOR.

Since VERTEX COLOR is not constant factor approximable, STRING PACK is not. ■

#### IV. EXPERIMENTAL RESULTS

This section presents experimental results by considering the morphology sensitive algorithm for call admission control of a number of video traces.

The traces used in the experiments were taken from [17]. Each video trace has different characteristics and bandwidth requirements ranging from talking head videos with low bandwidth demand and rate variability to high-action scene videos with much higher demand and variability. The characteristics of each video are shown in Table III. Each video trace has 40,000 frames (except from the news1 trace) captured at a frame rate of 25 frames/sec. This makes the total time for most video traces to correspond to approximately 26 minutes.

The segments are computed from each video trace by averaging each frame over the corresponding frame period. If additional bandwidth shaping/smoothing were to be performed for each video trace, the smoothed output would need to be used instead for the segments. Such shaping can be done using any of the work-ahead video smoothing algorithms found in the literature [8]–[11].

The experiments were conducted by running the morphology sensitive algorithm on all pairwise combinations of the video traces. The efficiency of the algorithm is measured by capturing the actual running time on a PC with a single Intel dual-core processor operating at 3.0 GHz with sufficient RAM (4 GB). We also measure the running time as a function of the available bandwidth, increasing which results in fewer intersection intervals that need to

TABLE III.  
CHARACTERISTICS OF THE VARIOUS TRACES USED TO VALIDATE THE MORPHOLOGY-SENSITIVE ALGORITHM.

Trace	# frames	Peak Rate (Kbps)	Average Rate (Kbps)	Rate Variance (Kbps)
MrBean	40,000	5,726.8	441.2	516.0
asterix	40,000	3,684.4	558.7	503.5
atp	40,000	4,771.4	547.2	510.2
bond	40,000	6,114.8	607.7	642.3
dino	40,000	2,990.8	326.9	368.7
lambs	40,000	3,355.6	182.7	279.8
mtv1	40,000	5,730.0	615.1	576.5
mtv2	40,000	6,285.2	494.5	536.3
news1	31,515	4,860.4	516.6	649.8
news2	40,000	4,747.2	383.9	487.6

TABLE IV.  
NUMBER OF INTERSECTION INTERVALS OF THE MORPHOLOGY SENSITIVE ALGORITHM FOR EACH PAIR OF VIDEO TRACES.

Trace	MrBean	asterix	atp	bond	dino	lambs	mtv1	mtv2	news1	news2
MrBean	182,291	81,531	249,185	185,982	42,064	42,375	323,808	289,023	391,415	140,293
asterix	81,531	7,939,968	1,345,016	123,995	2,269,795	802,838	237,921	318,873	1,841,422	785,606
atp	249,185	1,345,016	2,072,264	273,946	528,680	285,948	450,478	373,483	2,252,691	1,248,676
bond	185,982	123,995	273,946	477,292	57,365	47,763	336,124	508,148	420,448	166,505
dino	42,064	2,269,795	528,680	57,365	3,427,622	402,286	108,322	157,074	661,010	213,424
lambs	42,375	802,838	285,948	47,763	402,286	284,442	68,172	91,358	342,523	128,622
mtv1	323,808	237,921	450,478	336,124	108,322	68,172	565,014	437,379	613,354	273,665
mtv2	289,023	318,873	373,483	508,148	157,074	91,358	437,379	562,888	356,425	248,350
news1	391,415	1,841,422	2,252,691	420,448	661,010	342,523	613,354	356,425	2,925,885	1,543,442
news2	140,293	785,606	1,248,676	166,505	213,424	128,622	273,665	248,350	1,543,442	833,971

be considered and a smaller time displacement for the admitted video.

In the first set of experiments we generate sufficient scheduling conflicts by setting the available bandwidth to the maximum peak rate of the video trace pair. The first video trace is scheduled at time zero while the second one is to be scheduled at the minimum displacement from time zero, assuming that there is no capacity violation. This corresponds to Video-on-Demand (VoD) use cases where users request video traces approximately at the same time instant (denoted by zero in this experiment). The number of conflicts computed are reflected on the number of intersection intervals shown in Table IV.

The running time of the morphology sensitive algorithm for the various video trace pairs is shown in Table V. It is worth mentioning that when the number of conflicts is substantial (as in the case of asterix vs. asterix), the running time of the algorithm is fairly small (several seconds), while in all other cases the running time is practically negligible.

A second set of experiments was performed to demonstrate the sensitivity of the running time of the algorithm to the available bandwidth, given the fact that a lower

available bandwidth generates more intersection intervals that need to be considered. We chose to use the asterix-asterix video trace pair as a reference because it exhibited significant number of scheduling conflicts which resulted in the largest running time of the algorithm in the first set of experiments. The results of the asterix-asterix and another three video trace pairs are summarized in a set of graphs (Figs. 8-11). The results for the rest of the video pairs follow similar patterns and have been omitted due to space considerations.

In Fig. 8 one can observe a rapid drop in the running time of the algorithm as the available bandwidth increases. This owes to the fact that lower available capacity results in higher number of intersection intervals considered by the algorithm for the computation of the minimal displacement, as seen in Fig. 10.

It is worth noting that the algorithm resulted in fairly low running times across all experiments. This supports the practical use of the algorithm in an online setting, i.e., call-admission control of a new request for a video trace (e.g. in Video-on-Demand environments). Online, the algorithm can be run in an iterative manner to perform the admission control of each incoming video request,

TABLE V.  
RUNNING TIME IN SECONDS OF THE MORPHOLOGY SENSITIVE ALGORITHM FOR EACH PAIR OF VIDEO TRACES.

Trace	MrBean	asterix	atp	bond	dino	lambs	mtv1	mtv2	news1	news2
MrBean	0.344	0.161	0.482	0.364	0.110	0.110	0.644	0.611	0.761	0.302
asterix	0.168	28.78	3.800	0.241	5.801	2.509	0.537	0.756	6.670	1.994
atp	0.432	3.101	5.702	0.489	1,011	0.509	0.933	0.895	6.705	3.224
bond	0.332	0.224	0.537	0.890	0.168	0.114	0.667	1.274	0.815	0.323
dino	0.109	5.479	1.188	0.174	9.556	0.858	0.221	0.327	1.632	0.493
lambs	0.111	2.414	0.634	0.115	0.778	0.590	0.148	0.190	0.691	0.255
mtv1	0.606	0.420	0.914	0.646	0.203	0.145	1.219	1.017	1.281	0.551
mtv2	0.506	0.586	0.710	1.022	0.271	0.172	0.872	1.129	0.708	0.449
news1	0.726	4.486	5.662	0.787	1.313	0.596	1.200	0.784	7.921	4.103
news2	0.274	1.577	3.146	0.308	0.428	0.235	0.561	0.530	3.866	1.834

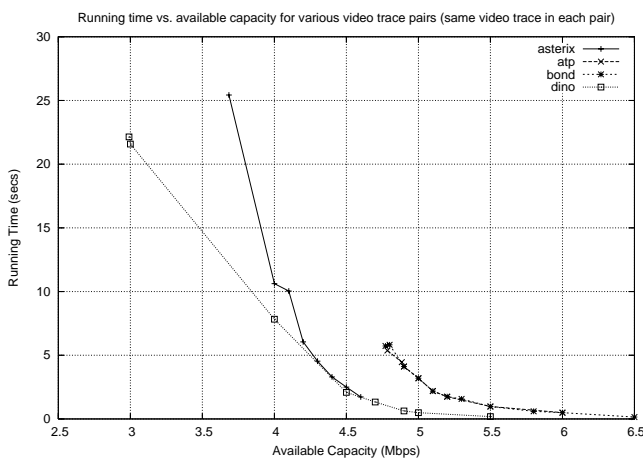


Figure 8. Running time vs. available capacity for admitting a new trace on top of another trace (same video trace in each trace pair).

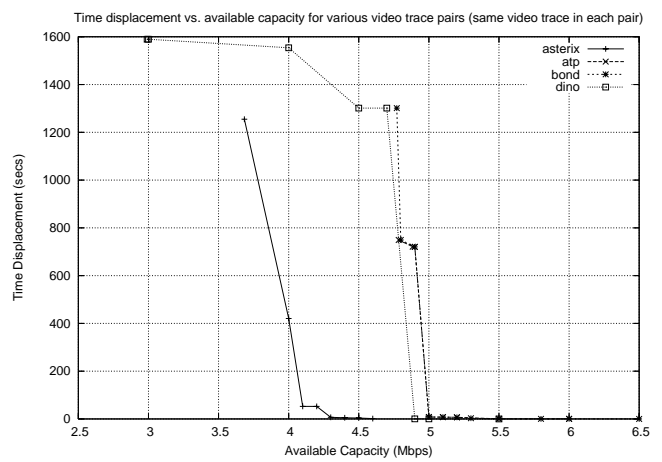


Figure 9. Time displacement vs. available capacity for admitting a new trace on top of another trace (same video trace in each trace pair).

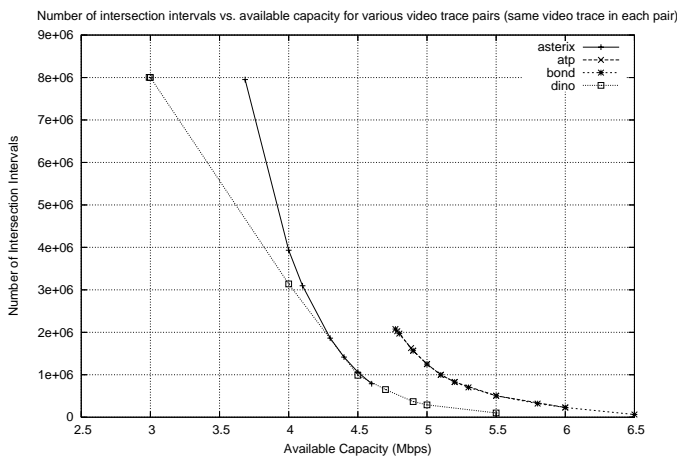


Figure 10. Number of intersection intervals vs. available capacity for admitting a new trace on top of another trace (same video trace in each trace pair).

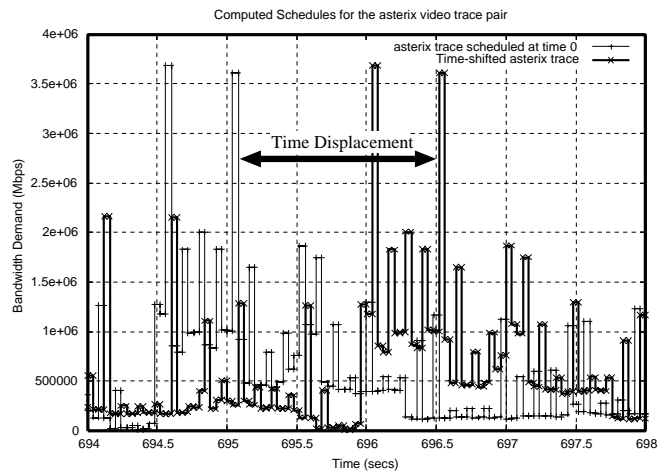


Figure 11. Example computed schedule after admitting a new asterix trace on top of another asterix trace with a total of 4.6 Mbps available capacity.

shaping a new traffic envelope before any subsequent video request.

Another important observation is that a very small

amount of additional capacity can be used to accommodate a new request at a very small time displacement (typically less than a second).

## V. CONCLUSIONS

In this paper, we examined the problem of accepting new video requests for pre-stored VBR video streams that have been pre-smoothed using any of the smoothing algorithms found in the literature. We proved that this problem is NP-complete and any constant approximation is NP-hard, by reducing it to the STRING PACK problem.

We also presented an algorithm that can be used to compute the minimum time displacement of a new request to avoid resource overbooking. The morphology-sensitive algorithm is capable of computing the time displacement in  $O((P+n)\log n)$  time complexity, where  $P = O(n^2)$  is the number of peak pairs that cause bandwidth over-allocation, when the first peak is selected from the schedule of the new request, and the second from the current traffic envelope, and  $n$  corresponds to the total number of peaks in the schedule of the new video request and the traffic envelope. We implemented and ran the algorithm on a variety of video trace pairs, demonstrating its effectiveness when used for video call-admission control in an online fashion, given its reasonably low running time.

This work can be extended in several ways. In particular, when the cost to the end-user is a variable that needs to be considered, and the cost is a function of the time displacement, the problem can be transformed into one that finds the minimal displacement at the minimally acceptable cost for the end-user.

Other optimization objectives could be analyzed, when given a set of requests, the requirement is to find the displacement points of the associated schedules that produce the smoothest combined schedule.

## REFERENCES

- [1] E. Casilari, A. R. Lecuona, A. D. Estrella, and F. Sandoval, "Classification and comparison of modelling strategies for vbr video traffic," in *Proceedings of International Teletraffic Congress (ITC-16) '99*, June 1999.
- [2] K. M. Elsayed and H. G. Perros, "On the effective bandwidth of arbitrary on/off sources," in *Proceedings of the Sixth IFIP WG6.3 Conference on Performance of Computer Networks*, October 1995, pp. 257–271.
- [3] A. I. Elwalid and D. Mitra, "Effective bandwidth of general markovian traffic sources and admission control of high speed networks," in *Proceedings of IEEE INFOCOM '93*, vol. 1, March 1993, pp. 256–265.
- [4] R. Guerin, H. Ahmadi, and M. Naghshineh, "Equivalent capacity and its application to bandwidth allocation in high-speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 968–981, September 1991.
- [5] G. Kesidis, "Modeling to obtain the effective bandwidth of a traffic source in an ATM network," in *Proceedings of the Second International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '94)*, Durham, NC, USA, January 1994, pp. 318–322.
- [6] J. Gao and I. Rubin, "Multifractal modeling of counting processes of long-range-dependent network traffic," 2001.
- [7] G. Li and V. Li, "Analysis of transient loss performance impact of long-range-dependence in network traffic," *Self-Similar Traffic and Performance Evaluation*, 2000.
- [8] W. Feng, *Buffering Techniques for Delivery of Compressed Video in Video-on-Demand Systems*. Kluwer Academic Publishers, 1997.
- [9] Z. Jiang and L. Kleinrock, "A general optimal smoothing algorithm," in *Proceedings of IEEE INFOCOM '98*, vol. 1, 1998.
- [10] J. M. McManus and K. W. Ross, "Video-on-demand over ATM: Constant-rate transmission and transport," *IEEE Journal on Selected Areas in Communications*, vol. 14, pp. 1087–1098, August 1996.
- [11] J. D. Salehi, Z. L. Zhang, J. F. Kurose, and D. Towsley, "Supporting stored video: Reducing rate variability and end-to-end resource requirements through optimal smoothing," in *Proceedings of ACM SIGMETRICS '96*, vol. 1, 1996.
- [12] J. Nickerson and S. Skiena, "Attention and Communication: Decision Scenarios for Teleoperating Robots," *IEEE. Proceedings of the Hawaii International Conference on System Sciences*, vol. January 3-6, 2005.
- [13] S. Skiena, *The Algorithm Design Manual*, 2nd ed. Springer, 2008.
- [14] A. Gajentaan and M. H. Overmars, "On a class of  $O(n^2)$  problems in computational geometry," *Comput. Geom. Theory Appl.*, vol. 5, pp. 165–185, 1995.
- [15] G. Barequet and S. Har-Peled, "Polygon-containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard," *Int. J. Comput. Geom.*, vol. 11, pp. 465–474, 2001.
- [16] U. Feige and J. Kilian, "Zero knowledge and the chromatic number," *J. Comput. System Sci.*, vol. 57, pp. 187–199, 1998.
- [17] "MPEG Traces." [Online]. Available: <http://www3.informatik.uni-wuerzburg.de/MPEG/traces/>

**Christos Tryfonas** received the Diploma in Computer Engineering and Informatics from the University of Patras, Greece, in 1993, and the M.S. and Ph.D. degrees in Computer Engineering from the University of California, Santa Cruz, in 1996 and 1999, respectively. He was a Principal Member of Technical Staff at Sprint Advanced Technology Laboratories, Burlingame, CA, until 2003. Currently, he is a co-founder and chief architect at Kazeon Systems, Mountain View, CA. His research interests lie in the areas of multimedia networking, Quality-of-Service (QoS), traffic management, storage networking, object storage, and network security in broadband networks.

**Dimitris Papamichail** is Assistant Professor of Computer Science at the University of Miami. He received his Diploma in Computer Engineering and Informatics from the University of Patras, Greece, in 1996, his M.S. in Computer Science from the University of Arizona in 1998 and his Ph.D. in Computer Science from Stony Brook University in 2007. His research interests include the design and implementation of applied algorithms and data structures, mainly in the area of bioinformatics.

**Andrew Mehler** was born in Long Island, New York. He received his Ph.D. in Computer Science from Stony Brook University in 2008. He also attended Cornell University, receiving a B.S. in computer science and electrical engineering in 2001, and an M.Eng in computer engineering in 2002.

**Steven Skiena** is Professor of Computer Science at Stony Brook University. His research interests include the design of graph, string, and geometric algorithms, and their applications (particularly to biology). He is the author of four books, including "The Algorithm Design Manual" and "Calculated Bets: Computers, Gambling, and Mathematical Modeling to Win". He is recipient of the ONR Young Investigator Award and the IEEE Computer Science and Engineering Undergraduate Teaching Award.

APPENDIX

*Self-Aligning Strings*

We now give a more precise account of Self-Aligning strings.

We call a set  $S = \{s_1 \dots s_n\}$  of strings  $(n, k, L)$ -aligning if the following properties hold.

- 1)  $|S| = n$
- 2)  $|s_i| = L$
- 3)  $\{s_i(0), s_j(0)\}$  is feasible, meaning no collision occurs
- 4)  $\{s_i(0), s_j(r)\}$  is not feasible for  $1 \leq r \leq L - k$



Figure A-1. Self aligning strings will either overlap completely (left) or overlap by some small, limited amount (right)

For a given  $(n, k, L)$ , there may or may not exist a set of self-aligning strings. We want to show a set exists that will make the reduction in the previous section work. That is, we need to be able to construct them in polynomial time (it is clear from construction that it takes  $O(nL)$  time to construct), and also we need certain constraints on  $n$ ,  $k$ , and  $L$ . The following two constraints are sufficient:

First, we want the ‘grouping’ effect. Thus our STRING-PACK strings should only be able to overlap by at most  $k$  (or equivalently only allow shifts of at least  $2L + \binom{n}{2} - k$ ). Since our STRING-PACK strings contain self-aligning strings as sub-strings, it is obvious that shifts of 1 to  $L - k$  are not allowed. Also, once we shift by  $\binom{n}{2} + k + 1$ , the prefix flanker of the shifted string overlaps the suffix flanker of the other string. Thus shifts of  $\binom{n}{2} + k + 1$  to  $2L + \binom{n}{2} - k$  are not allowed (Figure A-2). To make these 2 ranges overlap, we need

$$L - k \geq \binom{n}{2} + k + 1$$

The second constraint is to be able to recover the number of groups from the span of the solution. Since two strings will overlap completely only if their corresponding vertices are non-adjacent, we can recover a coloring by grouping strings that overlap completely. Let us assume the answer to STRING-PACK has  $C$  groups of strings that overlap completely. Since (from above) each can overlap at most  $k$ , this means the span of the solution is in the range

$$C * (|s| - k) + k \leq \text{span} \leq C * |s|$$

If the answer had  $C - 1$  groups, the range would be

$$(C - 1) * (|s| - k) + k \leq \text{span} \leq (C - 1) * |s|$$

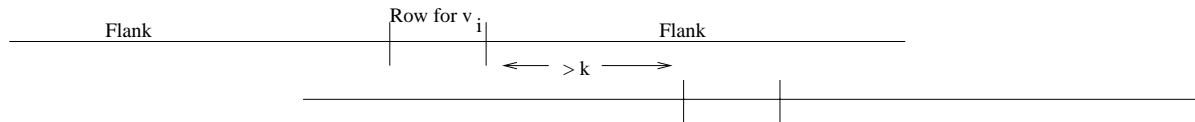


Figure A-2. The prefix copy of  $a_j$  needs to overlap at least  $k$  with the suffix copy of  $a_i$

To be able to distinguish the number of completely overlapping groups from the span, we would need

$$C * (|s| - k) + k > (C - 1) * |s|$$

That is, the smallest span from  $C$  groups is larger than the largest span from  $C - 1$  groups. This yields

$$|s| + k > k * C$$

Since  $C \leq n$ , this inequality is achieved if

$$|s| + k > kn$$

$$2L + \binom{n}{2} + k > kn$$

$$L > \frac{1}{2}k(n - 1) - \binom{n}{2}$$

These constraints are easy to achieve with the outlined construction. To be precise, our self-aligning strings are the rows of

$$[R_1 R_2 \dots R_n I^l P_{1,1} \dots P_{n,n}]$$

Where  $R_i$  is the  $nxn$  matrix

$$\begin{matrix} 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ n \end{matrix} \begin{pmatrix} 0000 \dots 0 \\ 0000 \dots 0 \\ \vdots \\ 1111 \dots 1 \\ \vdots \\ 0000 \dots 0 \end{pmatrix}$$

and  $P_{i,j}$  is the  $nxn^2$  matrix

$$\begin{matrix} 1 \\ 2 \\ \vdots \\ i \\ \vdots \\ j \\ \vdots \\ n \end{matrix} \begin{pmatrix} 0000 \dots 0 \\ 0000 \dots 0 \\ \vdots \\ 1000 \dots 0 \\ \vdots \\ 0000 \dots 0 \\ \vdots \\ 0111 \dots 1 \\ \vdots \\ 0000 \dots 0 \end{pmatrix}$$

The  $R_i$  give each row  $n$  consecutive 1s. Once a string is shifted right by  $n^2$ , its  $R_i$  will all lie in the  $I$  region of the other strings (that is, it will have  $n$  consecutive 1's overlapping the  $I$  matrices). Since in this  $I$  region a string has a 1 every  $n$  positions, this shift is not feasible (the  $n$  1s in the shifted string must conflict with a 1 in the other strings). Thus shifts of  $n^2$  through  $ln$  are not feasible ( $n^2$  ensures that an  $R_i$  is completely in the  $I$  regions;  $ln$  ensures an  $R_i$  is not shifted past the  $I$  region). The  $P_{i,j}$  eliminate shifts of 1 through  $n^2 - 1$ . This is done explicitly, as can be seen in the construction of the  $P_{i,j}$  matrices.

Shifts of 1 through  $ln$  are not feasible and thus the maximum overlap,  $k$ , is bounded by

$$k \leq L - ln = (n * |R_i| + l * |I| + n^2 * |P_{i,j}|) - ln = n^2 + ln + n^4 - ln$$

$$k \leq n^2 + n^4$$

This is a good result since  $k$  is fixed for any size  $l$ . All we want is  $L > \max(2k + \binom{n}{2}, \frac{nk}{2})$ , which we get for large enough  $n$  by setting  $l = n^4$ , which in turn gets  $L = n^2 + ln + n^4 = O(n^5)$ ; a polynomial length string, as desired.